Building a Simulator and Emulator for Traffic Signaling

Wenyu Li [*1] **Yilin Sun** [*2] **Xiangyu Zhao** [*3]

1. Introduction

The history of traffic signalling dates back to 1868, when the first traffic light was installed at the houses of parliament in London. Over the years, traffic planning began to show significant influence in various aspects. Inappropriate traffic planning results in congestion, leading to a waste of time and fuel and an increase in carbon emissions (Braess et al., 2005; Raymond, 2001). This makes traffic signalling not only an interesting problem but also an important challenge for countries to tackle.

In fact, traffic expands to a much broader definition. The Internet we use every day is a form of traffic. With packets continuously arriving at routers, the sequence these packets are processed and forwarded plays an important role in efficient networking and communication. By minimising congestion, we would be able to get shorter latency, higher throughput and lower packet drop rate, thus improving the quality of service (Thompson et al., 1997; Moore & Zuev, 2005). Broadly speaking, for most graph structures in our life, some planning is involved, be it supply chain (Min & Zhou, 2002), electric circuits (Calhoun et al., 2008) or telecommunication systems (Frost & Melamed, 1994). Therefore, the problem of traffic planning is indeed of great academic significance.

Computer modelling makes simulation and optimisation of traffic a real possibility. Conventionally, traffic modelling includes three critical parts: road layout, signal schedule, and traffic density (Hoogendoorn & Bovy, 2001). Simulation of the traffic can occur at different granularity and fidelity (Azlan & Rohani, 2018). Microscopic models regard each car as a separate element, while macroscopic models describe the traffic by density (Treiber et al., 2006; Lee et al., 2001). In between are hybrid approaches where vehicles are analysed in small groups (Burghout, 2004). In this project, we attempt to simulate traffic microscopically and use Bayesian optimisation (Močkus, 1975; Snoek et al., 2012) to emulate and optimise the signalling schedule.

[*]Equal contribution [1]Girton College, MPhil in ACS [2]Magdalene College, CST Part III [3]Trinity College, CST Part III. Correspondence to: Wenyu Li <wl414@cam.ac.uk>, Yilin Sun <ys512@cam.ac.uk>, Xiangyu Zhao <xz398@cam.ac.uk>.

2. Task Description

2.1. Network

A network, simply put, is a directed graph. Its junctions are vertices and its roads are edges. A two-way road can be considered as a pair of edges connecting the same pair of junctions, with the same length but opposite directions.

We also make the following assumption on all of the graphs we simulate:

Assumption 2.1. Each junction contains exactly two incoming roads.

While this seems a very bold assumption to be made, it is actually not, as any graph could be transformed into the form mentioned above without losing any property desired in the simulation.

However, two special cases in the network might still exist. Firstly, there can be two roads between the same pair of junctions in the same direction. Secondly, roads can originate and end at the same junction. Therefore, technically, our graph is not only directed, but also could be multi-edged and self-looped. However, we have also provided initialisation setting to prevent such properties upon random generation of the graph if the user desires.

At this point, you may wonder why do we have to make such an assumption. Furthermore, why do we not care about outgoing edges but only incoming edges? The answer is short – simplicity. To explain this, we would first need to look at how traffic is scheduled, which would give a natural explanation of the rationale behind.

2.2. Traffic Signal

The traffic light schedule, as one might expect, defines at each time point which roads should allow the release of queuing cars. This could be easily explained by Figure 1: when $T = 0$, traffic light is green for a-street. Therefore, the yellow car is dequeue and move onto d-street at $T = 1$. Similarly, the green car gets dequeued at $T = 1$. To simulate the real-life situation, we make the second assumption in this section:

Assumption 2.2. At any point, for every junction, exactly one road is allowed to release its queue of cars.

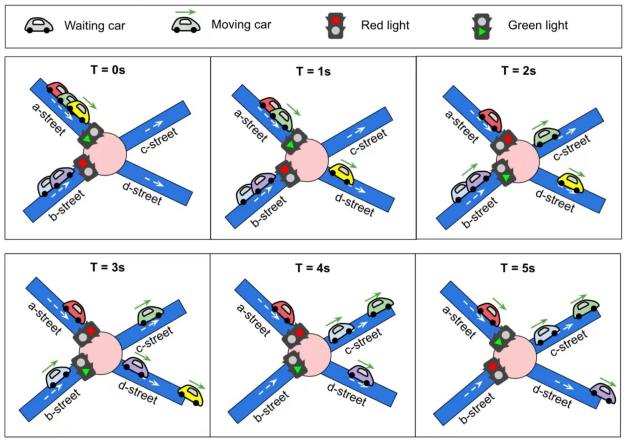


Figure 1. This figure shows how five cars queue at a junction and pass through according to the traffic signal (Google, 2021). The junction has two outgoing roads and two incoming roads. The green light for a-street lasts for 2 time steps so two of the three cars waiting at that street drive through and the other car has to wait for another round.

This avoids problems such as multiple roads releasing cars at the same time or no road releasing its cars. Certainly, we could define more sophisticated schedules, allowing multiple roads to release cars simultaneously, like a crossroad. However, that only complicates our simulation with little benefit added.

This clearly explains why we want every junction to have exact two incoming roads, simply because this is the most basic case we could have with a meaningful schedule. A junction with only one or no incoming road does not need scheduling to avoid cars crashing into each other. In addition, the outgoing road plays no part in the scheduling process as cars are passively added to them after they pass the junction and there is no competition happening between them.

Finally, throughout the simulation, we only use periodic schedule, which is a special kind of scheduling that periodically switches between red and green lights. Although admittedly a theoretically general traffic signal scheduling does not set constraints on periodicity, any properly functioning traffic signal in the real world would have a fixed duration for red and green lights. Therefore, to simulate the real-world scenario, we will focus on periodic schedules for all junctions in our setup.

2.3. Cars

Each car is regarded as an independent element and has its own travel plan when it gets deployed on the network. The travel plan of each car is a sequence of roads. The car has to travel along the designated roads without any detour until it reaches its destination, from when it would remain stationary until the end of the simulation.

Every car has a uniform speed of 1, meaning it takes L units of time to complete a road of length L. Crossing junctions produces no delay by assumption. When a car reaches the end of the road it is travelling on; it joins the queue (and gets immediately released if the light is green). The queue acts in a first-in-first-out (FIFO) fashion so the cars wait until their turns to move.

2.4. Reward

The final goal of our optimiser is to maximise a certain reward function. Formally, we define the reward to be the cumulative distance travelled by all cars within the system:

$$\text{Reward} = \sum_{c_i \in \text{Cars}} \text{Distance travelled by car } c_i$$

This is slightly different from the definition of the scoring function in the Hash Code competition (Google, 2021):

$$\text{Score for car } i = F + (D - T) \text{ if } T \leq D \text{ else } 0$$

where F is the bonus granted for finishing the route, T is the time when the car finishes and D is the total distance travelled by the car.

We choose to avoid the latter function because it is not analytic for optimisation. A slight change in traffic scheduling could easily make a number of cars unable to finish their route, thus setting their score to 0. This makes it extremely difficult to locate minimum points for the optimiser.

2.5. Transformation

We have glossed over the fact that network with purely 2-incoming junctions are general enough since any graph could be transformed into this form without losing the desired properties. Here we propose the following algorithm for transformation:

Case 1
if Junction j has 0 incoming edge **then**
  for i in range(2) **do**
    add self loop to j
  end for
end if

Case 2
if Junction j has 1 incoming edge e **then**
  duplicate e
end if

Case 3
if Junction j has k incoming edge e and $k > 2$ **then**
  nodes $\leftarrow j$.incoming

while $len(nodes) > 2$ **do**
$aux = node()$
$n1 = nodes.pop()$
$n2 = nodes.pop()$
del $edge(n1, j, w1),\ edge(n2, j, w2)$
add $edge(n1, aux, w1),\ edge(n2, aux, w2)$
add $edge(aux, j, 0)$
$nodes.add(aux)$
end while
end if

As a simple example, we could see how a 5-incoming junction could be transformed into 2-incoming junctions via the process shown in Figure 2.

So what property does this transformation preserve exactly? Firstly, it preserves the total distance between the nodes. This is important as the distance impacts the travel time of the car. Secondly, any schedule we make in the original graph, there would be a corresponding schedule in the transformed graph. Similarly, for any schedule in the transformed graph, there is a corresponding schedule in the original graph. This nice one-to-one correspondence allows us to take an arbitrary graph, transform it, optimise the schedule and transform the schedule back. Allowing our 2-incoming network to model any graph in general.

3. Simulation

To tackle the problem defined above, we first built a simulator that models the system and performs simulations based on given initialisation of the environments and inputs of traffic signal schedules. This enables us to find the optimal traffic signalling schedule via exhaustive search and Monte Carlo methods. More importantly, it provides the tool for data point generation in the emulator at the later stage.

3.1. System setup

In our simulator, we model the network and traffic using Object-oriented Programming techniques. Each `Network` object would consists of a list of junctions and a list of road, which have `Junction` and `Road` class defined respectively.

A `Junction` keeps track of incoming and outgoing roads as lists and it also keeps a `Schedule`. The `Schedule` class contains a function from time to index, representing the traffic signalling process to select the road for car release at certain point in time. We defined a child class of `Schedule` named `PeriodicSchedule`. It denotes a special schedule which selects the roads in a round-robin fashion, with good resemblance to the real-world scenario.

Finally, we defined the `Road` and `Car` classes to denote roads and cars respectively. `Road` keeps a queue that records the cars waiting at the junction while `Car` records the road it is on, the travel route to follow and the distance travelled so far. Once the car has completed its route, it stops at its final destination junction, and will not move again for the entire simulation.

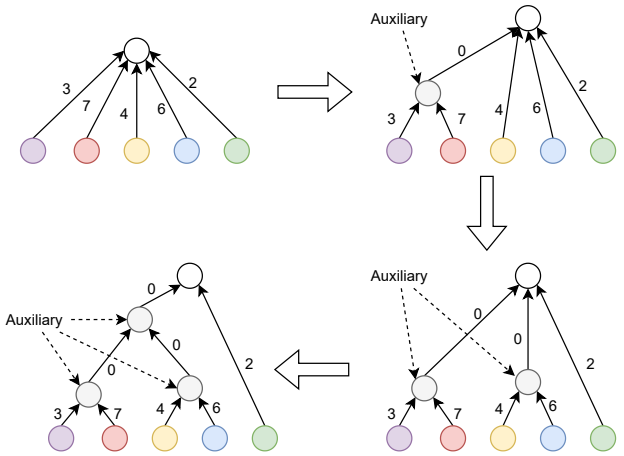


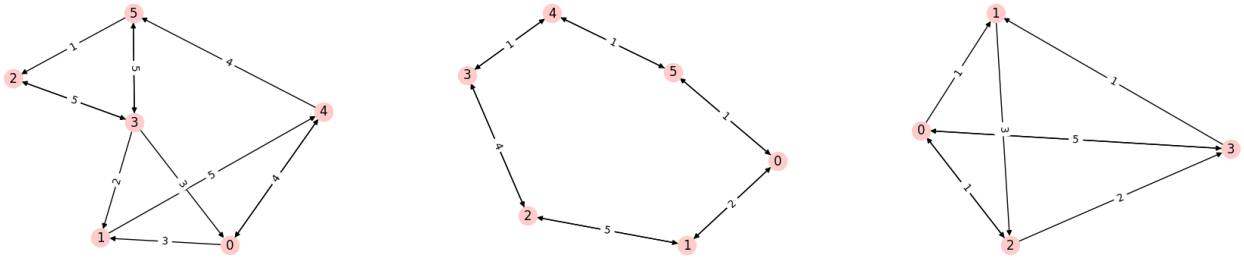
Figure 2. Transform from multiple incoming roads to two incoming roads.

3.2. Network and route generation

In our simulator, both the network structure and the route for each car can be initialised either by manual setup or random generation. For the random generation of the network structure, we introduced the following two alternatives:

Random connected network According to the assumptions of a network as described in Section 2.1, we need to generate a network in which the number of roads is exactly twice the number of junctions. In order to generate a network that has as much randomness as possible, while preserving adequate suitability for our problem, we have tried various random generation algorithms, but most of them poses the following two problems:

- Deadends: since there are only $2n$ roads in a network containing n junctions, the network is quite sparse, and most naïve random network generation algorithms can frequently generate a network that contains deadends, i.e. junctions with no outgoing roads. Deadends can cause any car reaching it to stuck, and prevents us from generating further route for the car, thus limiting the total length of a car's route.
- Clustering: a naïve random network generation algorithms also tends to generate a network that can be divided into multiple disjoint sub-networks, reducing the overall connectivity of the network. If a network does not provide enough connectivity, the cars in the



(a) Random connected network (b) Random ring network (c) Network from text input

Figure 3. Three simple network initialisations by our simulator using different methods. Note that in a random ring network, the roads with opposite directions connecting two junctions are allowed to have different lengths, as opposed to the example shown in (b).

network cannot have any meaningful route other than travelling around in cycles.

In order to resolve these two problems, we eventually designed an algorithm that generates a random network by splitting the junctions of the network into two disjoint subsets, one containing the already interconnecting junctions, and the other containing the isolate junctions. For each junction j, as long as both subsets are not empty, one junction is randomly sampled from each subset, and a new road originating from each of the sampled junctions to junction j is built. If one of the subsets is empty, then both junctions are sampled from the other subset, which can guarantee to have at least two different junctions. The lengths of the roads is also randomly generated, satisfying the constraint that $L \leq L_{\text{max}}$. This algorithm not only preserves sufficient randomness for the network it generates, but also avoids deadends and guarantees adequate connectivity.

Random ring network This is a special network structure that consists of a ring connecting all the junctions, with every pair of adjacent junctions interconnecting each other by two roads of opposite directions. The lengths of the roads are randomly generated, also satisfying $L \leq L_{\text{max}}$. The two roads with opposite directions connecting the two adjacent junctions are not required to have the same length. An interesting property of the network is its symmetry as all nodes in the graph have the same degree.

Network from text input In addition to the random generation of the network structures and the car routes as described above, we also supports the initialisation of the system via text input, adopting the input formatting of Google Hash Code 2021 Qualification Round (Google, 2021). Figure 3 shows the visualisations of some sample networks generated using each of the aforementioned methods, drawn using the `NetworkX` library (Hagberg et al., 2008).

3.3. Simulation mechanisms

In our simulator, under a given initialisation of the network and car routes together with a fixed input schedule for the traffic lights, we simulate the system within a preset duration in a time frame by time frame manner, using a mechanism called *ticking*: for every time frame, we take turns to simulate the action of every junction and every car in that exact time frame. At each time frame, a junction firstly picks an incoming road whose traffic light is green, which is determined by the traffic signalling schedule of that junction. If that incoming road still has cars waiting in its queue, it dequeues a car from the queue, and let the car advance to the next road in its preset route. Besides, at each time frame, a car whose route has not been completed either travels a distance of 1 on the middle of its current road, or is pushed into the road's queue if it reaches the end of the road. As long as a car travels within a time frame, it increments its reward by 1. At the end of simulation, the sum of rewards of all the cars is calculated, as the reward of the input schedule under the given system initialisation.

4. Emulation

In addition to the simulator, we also built an emulator to carry out automatic optimisation of the traffic light scheduling. We also collected data on different model choices to evaluate their effectiveness. In this section, we will elaborate our approach to the emulation, followed by a description of various models and heuristics behind. Lastly, we would explain about our data collection process for the emulation.

4.1. Emulation Pipeline

The general emulation process could be broken down into three different stages:

- scheduling: transform input X into list of schedules corresponding to each junction;
- simulation: build a simulator using the schedule design

from step above;

- Bayesian optimisation: use `GPyOpt` (GPy, since 2012; The GPyOpt authors, 2016) to carry out optimisation using Gaussian process.

One interesting observation about the pipeline is that stage 1 is where different model choices manifest, whereas stage 2 and 3 are almost identical throughout, meaning if we would like to benchmark the performance of different models, all we have to change is the scheduling part of our program.

4.2. Models

We now move on to elaborate on different model choices we have tested and the heuristics behind. As mentioned, the scheduling stage is the key to modelling, so we will focus on that part.

4.2.1. Distinct Scheduling

One natural scheduling choice is to assign two parameters to each junction for our input. This is because we have simplified our problem such that each junction contains exactly two incoming roads. This means two parameters: time for 'left' incoming and time for 'right' incoming are able to describe the scheduling completely.

However, while this model clearly provides the most detailed description of our state space, it is clearly the least efficient to emulate. This leads us to the problem of 'curse of dimensionality', a recurring yet difficult problem in machine learning. This is especially the case in bayesian optimisation, where the large number of parameters leads to long calculation time for the posterior distribution (Snoek et al., 2012). Consider the case where we have merely 40 junctions. This could easily give us 80 parameters to optimise. Assuming each parameter could take discrete values between 1 to 60, this gives 60^{80} possible states to explore, which could takes hours, if not days to get anything even close to optimal.

4.2.2. Uniform Scheduling

The infeasibility of high-dimensional optimisation leads to another extreme end of choice, which is to use the same schedule for every junction. This quickly reduces our number of parameters to merely 2.

This clearly gives us a much more efficient emulator. If time is severely constrained or prioritised (e.g. in a real-time system), this method could be a feasible technique. However, the advantage with time does not comes free as we have sacrificed flexibility. The descriptive power of this model is very limited as it does not account for the differences arising from network structure or car route. However, we found that it still performs decently well in networks with small scale or symmetries due to the small variability involved.

The advantages and disadvantages of this model are usually observed as fast convergence rate but lower convergence limit as we would soon discover in our experiments.

4.2.3. Preset Scheduling

Just like most design choices, the extremes usually are not the optimal choices. Seeking balance between both ends of the spectrum commonly brings the best of the two worlds. That is what leads to our third model choice, the preset scheduling.

As the name suggests, we use a few preset schedules and allow each junction to choose between them. This is in fact simply a hybrid between pure distinct and pure uniform. Unlike distinct scheduling, we are not giving each junction so much free that the size our state space gets out of control. However, different from uniform scheduling, we also do no constrain our choice too much that the power of our optimisation become too limited. Indeed, to strike a balance it is.

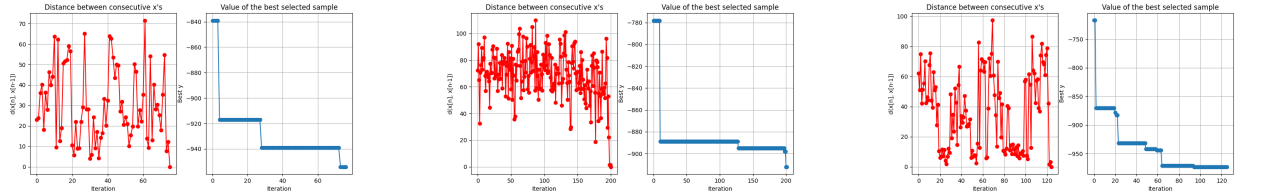
As such, we may describe our model formally as follows:

- the input X contains:
  - n preset schedules, described by n pairs of red/green duration: (r_1, g_1), (r_2, g_2), ..., (r_n, g_n), and
  - a schedule choice of j junctions (called *modes*), described by s_1, s_2, ..., s_j, where $1 \leq s_i \leq n$;
- we then construct our simulator, by assigning junction i with periodic schedule (r_{s_i}, g_{s_i});
- finally, we carry out Bayesian optimisation as usual.

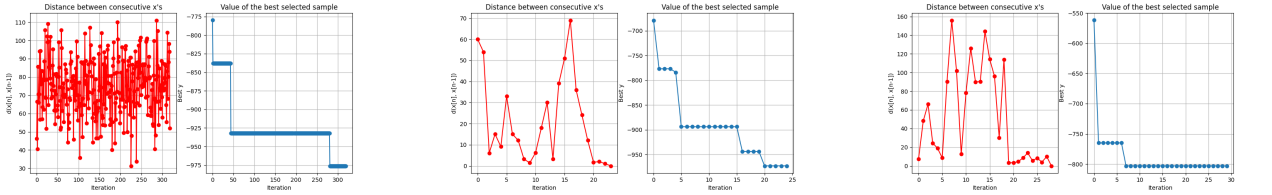
The number of parameters in this model is only $2n + j$, almost only a half of that compared to distinct scheduling when j is large. More importantly, the j mode choices are constrained to take only integer value from 1 to n. If we consider the case of 40 junctions, 3 modes, maximum duration of 60 seconds for each incoming road, the distinct scheduling gives a input space of $60^{80} \approx 1.8 \times 10^{142}$ while the preset scheduling gives only $60^3 \times 3^{40} \approx 2.6 \times 10^{24}$, which is indeed a significant reduction in search space.

4.2.4. Two-staged Preset

However, even for something in the scale of 10^{24}, it could still be a great challenge for our optimiser. A more serious problem associated with the above method is that it does not scale. As the number of junctions increases, the number of parameter grows with it linearly and the search space grows exponentially. This gives a frustrating resemblance to NP-complete problems, which is almost impractical to get close-to-optimal solutions in polynomial time.



(a) Uniform scheduling (b) Distinct scheduling (c) Preset scheduling with 2 modes



(d) Preset scheduling with 5 modes (e) Forced-preset scheduling with 2 modes (f) Forced-preset scheduling with 5 modes

Figure 4. Convergence graph for running Bayesian optimisation on a 5-junction network

Schedule	Opt. schedule	Opt. reward	Convergence iter.
Uniform	[17, 12]	931	78
Distinct	[(21, 38), (30, 41), (26, 13), (32, 29), (26, 36)]	964	200
Preset (2 modes)	schedules: [(16, 29), (54, 9)] modes: [0, 0, 0, 1, 0]	**1014**	125
Preset (5 modes)	schedules: [(15, 13), (40, 47), (31, 17), (14, 7), (29, 31)] modes: [0, 2, 2, 2, 0]	984	322
Forced-preset (2 modes)	cycle length: 6 modes: [0, 0, 0, 1, 1]	**992**	**24**
Forced-preset (5 modes)	cycle length: 48 modes: [1, 2, 2, 3, 3]	939	**29**

Table 1. Experimental results for running Bayesian optimisation on a 5-junction network

To tackle this problem, we would like to have a fixed or logarithmic number of parameters with respect to the number of junctions. We believe the two-staged preset model might be able to offer an interesting solution.

We believe that the schedule choice is very closely linked to the nature of the network, assuming the car routes are distributed randomly enough. Consider in real life, we often find some preference on time allocation for different incoming roads. Simply look at the junction outside the Computer Laboratory, the traffic light allocates much more time for Madingley Road compared to JJ Thompson Avenue simply because the former enjoys much greater traffic flow. The reason that Madingley Road has a greater traffic flow is due to the fact that it is a trunk road, connecting the city center to other central locations, both being nodes of high degrees.

Such observations would mean that given a road network, we might be able to pre-train a specific scheme even without any traffic and fine tune the parameters given specific car routes. This allows us to devise the following two-staged algorithm:

```
# Stage 1 (Pre-training):
# network is taken from input
# cars are generated at random
# schemes are fixed to cover the cases
    where favour 'left'/'right' or fair
# choices are subjected to optimisation

network = parse_network_from_input()
cars = generate_random_cars()
schemes = [[1,3], [2,2], [3,1]]
domain = {...scheme choices for junction
    in junctions...}
opt.optimize(
    lambda choices: simulator.simulate(
        network, cars, schemes, choices
    ), domain)

# Stage 2 (Fine-tuning):
# network same as stage 1
# cars are taken from input
# schemes are subjected to optimisation
# choices are fixed
```

```
network = parse_network_from_input()
cars = parse_cars_from_input()
choices = opt.opt_x
domain = {...schemes for junction
    in junctions...}
opt.optimize(
    lambda scheme: simulator.simulate(
        network, cars, schemes, choices
    ), domain)
```

While significant cost in pre-training is still being incurred, the fine-tuning time could be drastically decreased as the number of parameters are fixed during the fine-tuning process. This is great in real life as the underlying road infrastructure rarely changes but the traffic running on it are subjected to frequent fluctuations.

Unfortunately, given the limited time frame, we were only able to implement the 2-staged preset algorithm but not benchmark it against others. However, the ideas proposed in this report could serve as an inspiration for interested readers to carry out extensions.

4.3. Data Collection

With the models being built and emulator being constructed, we then moved on to our data collection process. For data collection, we mostly rely on the in-built functions: `plot_acquisition` and `plot_convergence` from `GPyOpt`. We observe trends from the graph and derive conclusions on the effectiveness of our emulation techniques.

For small scale problems, we were able to get the ground truth, i.e., the most optimal solution from exhaustive search. However, as problem size gets larger, exhaustive search quickly becomes infeasible and therefore comparison is based purely on convergence speed and final convergence limit. Thus, for large scale problems, the comparisons are purely relative between each method with no conclusive result on individual effectiveness as of how close to ground truth it gets.

5. Experiments

In this section, we will describe the Bayesian optimisation experiments we performed using different signalling models, and evaluate their results as well as comparing their performance in terms of convergence speed and limit.

5.1. Experimental Setup

We have carried out extensive trial and test on our model on problems of varying scale. In particular, we have run our emulator on junction size of 5, 40, 200 respectively to emulate networks of small and medium scale. We also aimed to run our emulator on junction size of 1000. However, we discovered that the number of parameter is too large to get any meaningful result in the given period of time. Therefore, our following discussion will be primarily focused on small and medium networks and potentially serve as a humble inspiration for problem of more realistic scale.

For each testing environment, we have used four different emulation techniques, namely *uniform* scheduling, *distinct* scheduling, *preset* scheduling, and *forced-preset* scheduling. The last emulation technique is based on the preset scheduling, but we assert the preset schedules to span a broad spectrum. For example, when the number of preset schedules is 3, we will use (15, 45), (30, 30), (45, 15) as our preset schedules when the cycle length is 60.

It should also be noted that the preset and forced-preset scheduling are run twice with the number of preset schedule being set to 2 and 5 respectively. This is to investigate the impact of schedule number on the effectiveness of these methods.

5.2. Results and Analysis

As mentioned in the previous subsection, the experiment is carried out on problems of different scale. As such, our discussion of the results will be split into the case where $n = 5,\ 40,\ 200$ respectively.

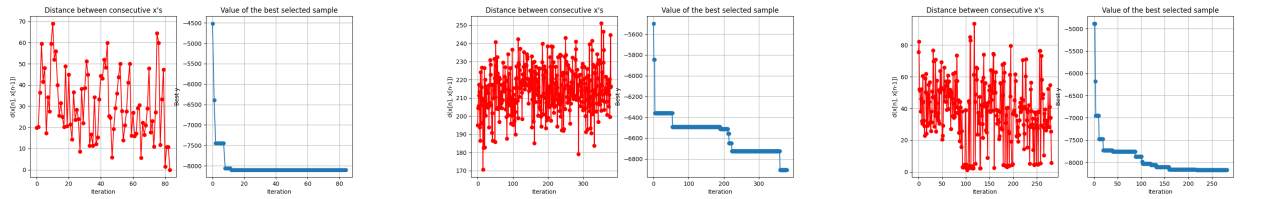
5.2.1. Very small network ($N = 5$)

On a very small network, we would be able to run all experiments until they fully converge. This gives us some meaningful results to compare the convergence limit. We have run the experiment multiple times on different seeds and select the result that is the most representative across all the trials conducted. The convergence graph and experimental results are attached in Figure 4 and Table 1.

Due to the small network size, the convergence of all 6 experiments are very close to each other. However, some minor difference could still be observed. In particular, the uniform scheme usually performs the worst out of all 6 experiments. This is due to the fact that the descriptiveness of merely 2 parameters is very limited. This prevents the model to capture differences in junctions and make changes to accommodate them.

The preset, forced-preset and pure distinct generally performs better with very close convergence limits, with preset usually performs the best out of the three. While this result seems quite surprising as the distinct model should capture more details and cover cases generated by the preset schedule, we believe the deterioration in performance could be explained by the following:

- The search space of pure distinct scheduling is way too big. This could make the optimiser land in local minimum when global minimum is too hard to locate.



(a) Uniform scheduling (b) Distinct scheduling (c) Preset scheduling with 2 modes

(d) Preset scheduling with 5 modes (e) Forced-preset scheduling with 2 modes (f) Forced-preset scheduling with 5 modes

Figure 5. Convergence graph for running Bayesian optimisation on a 40-junction network

Schedule	Uniform	Distinct	Preset (2)	Preset (5)	Forced-preset (2)	Forced-preset (5)
Opt. reward	8096	6904	8164	**8176**	**8175**	7320
Convergence iter.	**83**	391	285	117	**88**	112

Table 2. Experimental Results for Bayesian optimisation on a 40-junction network

- Distinct scheduling has a particularly slow convergence rate. This makes it especially tricky to decide when we have reached our limit.

On the convergence rate, without any surprise, uniform scheduling converges in only a few iterations, thanks to its small number of parameters. While preset usually converges around 100 iterations and 300 iterations when mode number is set to 5. Forced-preset also converges extremely fast, due to the small search space. Finally, distinct has a relatively slow convergence rate, taking almost 200 iterations to converge. This fits well with our expected results.

5.2.2. Small network ($N = 40$)

In the small network, we increase the number of junctions and cars to 40 and 400 respectively. Four scheduling methods are used, and the number of modes in preset and forced-preset scheduling is set to 5 and 2 for comparison. Also, the gap among solutions found by four scheduling methods enlarges with a larger network.

The force-preset scheduling with 5 modes finds the solution with the best reward of 8176, which outperforms all other scheduling schemes. The results obtained by preset and force-preset scheduling with 2 modes are similar and a bit higher than uniform scheduling result. However, due to the increase in network size and vast search space, the distinct scheduling and force-preset scheduling with 5 modes provide solutions with much lower rewards.

Based on the results, we have following findings:

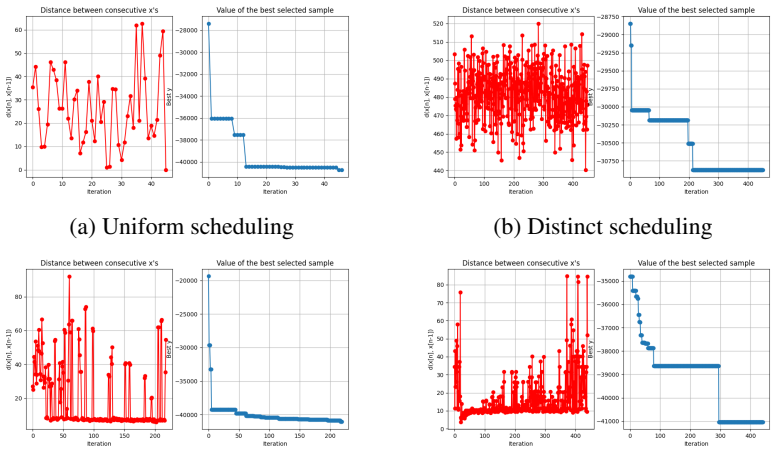
- Preset (2 and 5 modes) and forced-preset (2 modes) scheduling find the solution with much higher rewards. The reason behind is that they allow more flexibility in scheduling compared with uniform scheduling, while limit the number of parameters compared with distinct scheduling.
- Increasing the number of modes to 5 leads to lower convergence rate. For forced-preset scheduling, the reward of final solution falls to below 8000 due to vast search space in this case.

5.2.3. Medium network ($N = 200$)

The network size is further increased with $N = 200$ and 2000 cars to narrow the gap between the emulation and real-world scenarios. In terms of the optimal reward, preset and forced-preset scheduling provide a better solution than the uniform scheduling as shown in Figure 6 and Table 3. Regarding convergence rate, uniform scheduling takes the least iteration of 47 to reach convergence, while distinct and force-preset scheduling take approximately 10 times of that.

Comparing with the results from small networks, we have the following findings:

- Medium network makes most of the schedules slower to reach convergence.



(a) Uniform scheduling

(b) Distinct scheduling

(c) Preset scheduling with 2 modes

(d) Forced-preset scheduling with 2 modes

Figure 6. Convergence graph for running Bayesian optimisation on a 200-junction network

Schedule	Uniform	Distinct	Preset (2)	Forced-preset (2)
Opt. reward	40705	30873	**41062**	**41030**
Convergence iter.	**45**	475	**218**	432

Table 3. Experimental Results for Bayesian optimisation on a 200-junction network

- The gap between uniform and distinct scheduling rewards increases remarkably. This is because distinct scheduling becomes more difficult to optimise with complex networks.
- Preset and forced-preset scheduling generally find better solutions than uniform scheduling in all network sizes.

6. Conclusion

In this project, we carried out simulation and emulation of an urban traffic signalling system. We first built a simulator to test how different signal scheduling choices affect the total distance travelled by cars in a given period. Based on this, we built an emulator to search for optimal scheduling using Bayesian optimisation. To overcome the problem of exploding search space without sacrificing flexibility or descriptiveness, we introduced four different scheduling schemes and conducted experiments to compare their performances. Results show that preset scheduling gives the best convergence limit under reasonable number of iterations in most cases. On the other hand, uniform and distinct scheduling gives much poorer performance due to their own limitations. Forced-preset does show certain potential in some cases, but is rather unstable compared to pure preset.

We believe that this simple investigation could inspire much more sophisticated research in this area. Some future extension would include:

- benchmarking and improving the 2-staged preset algorithm;
- improving the randomness of graph generation;
- finding other scalable and efficient optimizations for real-world networks;
- integrating the transformation mentioned in Section 2.5 into the optimisation pipeline to tackle problems in a more general setting.

We believe the combination of Bayesian optimisation with traffic planning offers some novel insights and has much more potential to be discovered. With sufficient research effort, this area would bring great benefit to city planners and the general public, with potential applications in other areas involving network traffic controls.

References

Azlan, N. N. N. and Rohani, M. M. Overview of application of traffic simulation model. In *MATEC Web of Conferences*, volume 150, pp. 03006. EDP Sciences, 2018.

Braess, D., Nagurney, A., and Wakolbinger, T. On a paradox of traffic planning. *Transportation science*, 39(4):446–450, 2005.

Burghout, W. *Hybrid microscopic-mesoscopic traffic simulation*. PhD thesis, KTH, 2004.

Calhoun, B. H., Cao, Y., Li, X., Mai, K., Pileggi, L. T., Rutenbar, R. A., and Shepard, K. L. Digital circuit design challenges and opportunities in the era of nanoscale CMOS. *Proceedings of the IEEE*, 96(2):343–365, 2008.

Frost, V. and Melamed, B. Traffic modeling for telecommunications networks. *IEEE Communications Magazine*, 32 (3):70–81, 1994. doi: 10.1109/35.267444.

Google. Hash Code 2021 Online Qualification Round: Traffic signaling. https://codingcompetitions.withgoogle.com/hashcode/archive, 2021. Last accessed on 17 January 2022.

GPy. GPy: A gaussian process framework in python. http://github.com/SheffieldML/GPy, since 2012.

Hagberg, A. A., Schult, D. A., and Swart, P. J. Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J. (eds.), *Proceedings of the 7th Python in Science Conference*, pp. 11–15, Pasadena, CA USA, 2008.

Hoogendoorn, S. P. and Bovy, P. H. L. State-of-the-art of vehicular traffic flow modelling. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 215:283–303, 2001.

Lee, H., Lee, H.-W., and Kim, D. Macroscopic traffic models from microscopic car-following models. *Physical Review E*, 64(5):056126, 2001.

Min, H. and Zhou, G. Supply chain modeling: past, present and future. *Computers & industrial engineering*, 43(1-2): 231–249, 2002.

Močkus, J. On Bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pp. 400–404. Springer, 1975.

Moore, A. W. and Zuev, D. Internet traffic classification using Bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 50–60, 2005.

Raymond, J.-F. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies*, pp. 10–29. Springer, 2001.

Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

The GPyOpt authors. GPyOpt: A bayesian optimization framework in python. http://github.com/SheffieldML/GPyOpt, 2016.

Thompson, K., Miller, G., and Wilder, R. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11(6): 10–23, 1997. doi: 10.1109/65.642356.

Treiber, M., Kesting, A., and Helbing, D. Delays, inaccuracies and anticipation in microscopic traffic models. *Physica A: Statistical Mechanics and its Applications*, 360(1):71–88, 2006.